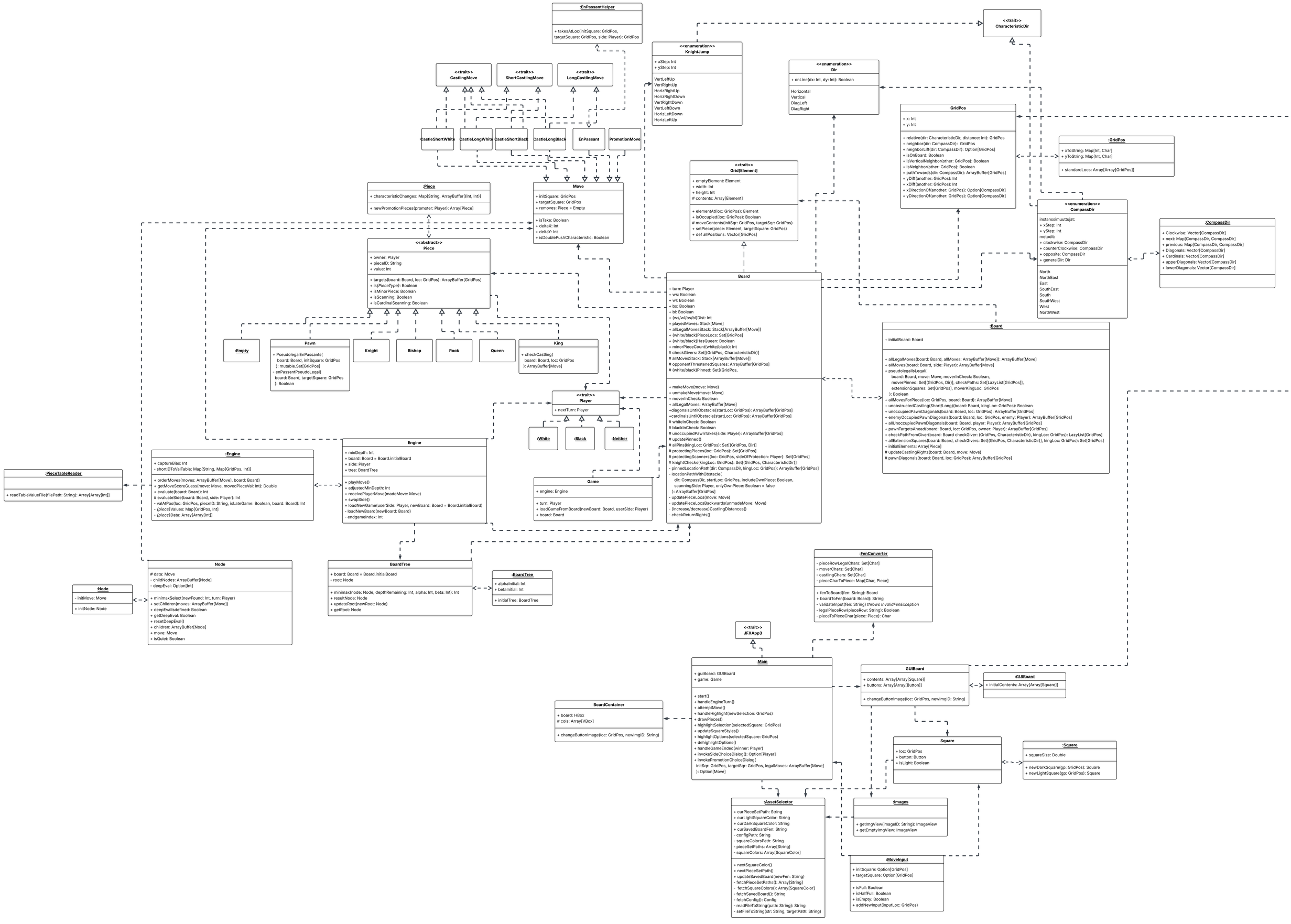
:EnPassantHelper
+ takesAtLoc(initSquare: GridPos, targetSquare: GridPos, side: Player): GridPos
<<trait>> CharacteristicDir
<<enumeration>> KnightJump
+ xStep: Int
+ yStep: Int
VertLeftUp
VertRightUp
HorizRightUp
HorizRightDown
VertRightDown
VertLeftDown
HorizLeftDown
HorizLeftUp
<<enumeration>> Dir
+ onLine(dx: Int, dy: Int): Boolean
Horizontal
Vertical
DiagLeft
DiagRight
<<trait>> CastlingMove
<<trait>> ShortCastlingMove
<<trait>> LongCastlingMove
CastleShortWhite
CastleLongWhite
CastleShortBlack
CastleLongBlack
EnPassant
PromotionMove
GridPos
+ x: Int
+ y: Int
+ relative(dir: CharacteristicDir, distance: Int): GridPos
+ neighbor(dir: CompassDir): GridPos
+ neighborLift(dir: CompassDir): Option[GridPos]
+ isOnBoard: Boolean
+ isVerticalNeighbor(other: GridPos): Boolean
+ isNeighbor(other: GridPos): Boolean
+ pathTowards(dir: CompassDir): ArrayBuffer[GridPos]
+ yDiff(another: GridPos): Int
+ xDiff(another: GridPos): Int
+ xDirectionOf(another: GridPos): Option[CompassDir]
+ yDirectionOf(another: GridPos): Option[CompassDir]
:GridPos
+ xToString: Map[Int, Char]
+ yToString: Map[Int, Char]
+ standardLocs: Array[Array[GridPos]]
:Piece
+ characteristicChanges: Map[String, ArrayBuffer[(Int, Int)]]
+ newPromotionPieces(promoter: Player): Array[Piece]
Move
+ initSquare: GridPos
+ targetSquare: GridPos
+ removes: Piece = Empty
+ isTake: Boolean
+ deltaX: Int
+ deltaY: Int
+ isDoublePushCharacteristic: Boolean
<<trait>> Grid[Element]
+ emptyElement: Element
+ width: Int
+ height: Int
# contents: Array[Element]
+ elementAt(loc: GridPos): Element
+ isOccupied(loc: GridPos): Boolean
# moveContents(initSqr: GridPos, targetSqr: GridPos)
+ setPiece(piece: Element, targetSquare: GridPos)
+ def allPositions: Vector[GridPos]
<<enumeration>> CompassDir
instanssimuuttujat:
+ xStep: Int
+ yStep: Int
metodit:
+ clockwise: CompassDir
+ counterClockwise: CompassDir
+ opposite: CompassDir
+ generalDir: Dir
North
NorthEast
East
SouthEast
South
SouthWest
West
NorthWest
:CompassDir
+ Clockwise: Vector[CompassDir]
+ next: Map[CompassDir, CompassDir]
+ previous: Map[CompassDir, CompassDir]
+ Diagonals: Vector[CompassDir]
+ Cardinals: Vector[CompassDir]
+ upperDiagonals: Vector[CompassDir]
+ lowerDiagonals: Vector[CompassDir]
<<abstract>> Piece
+ owner: Player
+ pieceID: String
+ value: Int
+ targets(board: Board, loc: GridPos): ArrayBuffer[GridPos]
+ is(PieceType): Boolean
+ isMinorPiece: Boolean
+ isScanning: Boolean
+ isCardinalScanning: Boolean
Board
+ turn: Player
+ ws: Boolean
+ wi: Boolean
+ bs: Boolean
+ bl: Boolean
+ (ws/wi/bs/bl)Dist: Int
+ playedMoves: Stack[Move]
+ allLegalMovesStack: Stack[ArrayBuffer[Move]]
+ (white/black)PieceLocs: Set[GridPos]
+ (white/black)HasQueen: Boolean
+ minorPieceCount(white/black): Int
# checkGivers: Set[(GridPos, CharacteristicDir)]
# allMovesStack: Stack[ArrayBuffer[Move]]
# opponentThreatenedSquares: ArrayBuffer[GridPos]
# (white/black)Pinned: Set[(GridPos,
+ makeMove(move: Move)
+ unmakeMove(move: Move)
+ moverInCheck: Boolean
+ allLegalMoves: ArrayBuffer[Move]
+diagonalsUntilObstacle(startLoc: GridPos): ArrayBuffer[GridPos]
+cardinalsUntilObstacle(startLoc: GridPos): ArrayBuffer[GridPos]
whiteInCheck: Boolean
blackInCheck: Boolean
unoccupiedPawnTakes(side: Player): ArrayBuffer[GridPos]
updatePinned()
allPins(kingLoc: GridPos): Set[(GridPos, Dir)]
protectingPieces(loc: GridPos): Set[GridPos]
protectingScanners(loc: GridPos, sideOfProtection: Player): Set[GridPos]
knightChecks(kingLoc: GridPos): Set[(GridPos, CharacteristicDir)]
- pinnedLocationPath(dir: CompassDir, kingLoc: GridPos): ArrayBuffer[GridPos]
- locationPathWithObstacle(
dir: CompassDir, startLoc: GridPos, includeOwnPiece: Boolean,
scanningSide: Player, onlyOwnPiece: Boolean = false
): ArrayBuffer[GridPos]
- updatePieceLocs(move: Move)
- updatePieceLocsBackwards(unmadeMove: Move)
- (increase/decrease)CastlingDistances()
- checkReturnRights()
:Board
+ initialBoard: Board
+ allLegalMoves(board: Board, allMoves: ArrayBuffer[Move]): ArrayBuffer[Move]
+ allMoves(board: Board, side: Player): ArrayBuffer[Move]
+ pseudolegalIsLegal(
board: Board, move: Move, moverInCheck: Boolean,
moverPinned: Set[(GridPos, Dir)], checkPaths: Set[LazyList[GridPos]],
extensionSquares: Set[GridPos], moverKingLoc: GridPos
): Boolean
+ allMovesForPiece(loc: GridPos, board: Board): ArrayBuffer[Move]
+ unobstructedCastling(Short/Long)(board: Board, kingLoc: GridPos): Boolean
+ unoccupiedPawnDiagonals(board: Board, loc: GridPos): ArrayBuffer[GridPos]
+ enemyOccupiedPawnDiagonals(board: Board, loc: GridPos, enemy: Player): ArrayBuffer[GridPos]
+ allUnoccupiedPawnDiagonals(board: Board, player: Player): ArrayBuffer[GridPos]
+ pawnTargetsAhead(board: Board, loc: GridPos, owner: Player): ArrayBuffer[GridPos]
+ checkPathFromGiver(board: Board checkGiver: (GridPos, CharacteristicDir), kingLoc: GridPos): LazyList[GridPos]
+ allExtensionSquares(board: Board, checkGivers: Set[(GridPos, CharacteristicDir)], kingLoc: GridPos): Set[GridPos]
+ initialElements: Array[Piece]
# updateCastlingRights(board: Board, move: Move)
# pawnDiagonals(board: Board, loc: GridPos): ArrayBuffer[GridPos]
:Empty
Pawn
+ PseudolegalEnPassants(
board: Board, initSquare: GridPos
): mutable.Set[GridPos]
- enPassantPseudoLegal(
board: Board, targetSquare: GridPos
): Boolean
Knight
Bishop
Rook
Queen
King
+ checkCastling(
board: Board, loc: GridPos
): ArrayBuffer[Move]
<<trait>> Player
+ nextTurn: Player
:White
:Black
:Neither
:PieceTableReader
+ readTableValueFile(filePath: String): Array[Array[Int]]
:Engine
+ captureBias: Int
- shortIDToValTable: Map[String, Map[GridPos, Int]]
+ orderMoves(moves: ArrayBuffer[Move], board: Board)
+ getMoveScoreGuess(move: Move, movedPieceVal: Int): Double
+ evaluate(board: Board): Int
# evaluateSide(board: Board, side: Player): Int
- valAtPos(loc: GridPos, pieceID: String, isLateGame: Boolean, board: Board): Int
- (piece)Values: Map[GridPos, Int]
- (piece)Data: Array[Array[Int]]
Engine
+ minDepth: Int
+ board: Board = Board.initialBoard
+ side: Player
+ tree: BoardTree
+ playMove()
+ adjustedMinDepth: Int
+ receivePlayerMove(madeMove: Move)
+ swapSide()
+ loadNewGame(userSide: Player, newBoard: Board = Board.initialBoard)
- loadNewBoard(newBoard: Board)
- endgameIndex: Int
Game
+ engine: Engine
+ turn: Player
+ loadGameFromBoard(newBoard: Board, userSide: Player)
+ board: Board
Node
data: Move
- childNodes: ArrayBuffer[Node]
- deepEval: Option[Int]
+ minimaxSelect(newFound: Int, turn: Player)
+ setChildren(moves: ArrayBuffer[Move])
+ deepEvalIsdefined: Boolean
+ getDeepEval: Boolean
+ resetDeepEval()
+ children: ArrayBuffer[Node]
+ move: Move
+ isQuiet: Boolean
BoardTree
+ board: Board = Board.initialBoard
- root: Node
+ minimax(node: Node, depthRemaining: Int, alpha: Int, beta: Int): Int
+ resultNode: Node
+ updateRoot(newRoot: Node)
+ getRoot: Node
:BoardTree
+ alphaInitial: Int
+ betaInitial: Int
+ initialTree: BoardTree
:Node
- initMove: Move
+ initNode: Node
:FenConverter
- pieceRowLegalChars: Set[Char]
- moverChars: Set[Char]
- castlingChars: Set[Char]
- pieceCharToPiece: Map[Char, Piece]
+ fenToBoard(fen: String): Board
+ boardToFen(board: Board): String
- validateInput(fen: String) throws InvalidFenException
- legalPieceRow(pieceRow: String): Boolean
- pieceToPieceChar(piece: Piece): Char
<<trait>> JFXApp3
:Main
+ guiBoard: GUIBoard
+ game: Game
+ start()
+ handleEngineTurn()
+ attemptMove()
+ handleHighlight(newSelection: GridPos)
+ drawPieces()
+ highlightSelection(selectedSquare: GridPos)
+ updateSquareStyles()
+ highlightOptions(selectedSquare: GridPos)
+ dehighlightOptions()
+ handleGameEnded(winner: Player)
+ invokeSideChoiceDialog(): Option[Player]
+ invokePromotionChoiceDialog(
initSqr: GridPos, targetSqr: GridPos, legalMoves: ArrayBuffer[Move]
): Option[Move]
GUIBoard
+ contents: Array[Array[Square]]
+ buttons: Array[Array[Button]]
+ changeButtonImage(loc: GridPos, newImgID: String)
:GUIBoard
+ initialContents: Array[Array[Square]]
BoardContainer
+ board: HBox
# cols: Array[VBox]
+ changeButtonImage(loc: GridPos, newImgID: String)
Square
+ loc: GridPos
+ button: Button
+ isLight: Boolean
:Square
+ squareSize: Double
+ newDarkSquare(gp: GridPos): Square
+ newLightSquare(gp: GridPos): Square
:AssetSelector
+ curPieceSetPath: String
+ curLightSquareColor: String
+ curDarkSquareColor: String
+ curSavedBoardFen: String
- configPath: String
- squareColorsPath: String
- pieceSetPaths: Array[String]
- squareColors: Array[SquareColor]
+ nextSquareColor()
+ nextPieceSetPath()
+ updateSavedBoard(newFen: String)
- fetchPieceSetPaths(): Array[String]
- fetchSquareColors(): Array[SquareColor]
- fetchSavedBoard(): String
- fetchConfig(): Config
- readFileToString(path: String): String
- setFileToString(str: String, targetPath: String)
:Images
+ getImgView(imageID: String): ImageView
+ getEmptyImgView: ImageView
:MoveInput
+ initSquare: Option[GridPos]
+ targetSquare: Option[GridPos]
+ isFull: Boolean
+ isHalfFull: Boolean
+ isEmpty: Boolean
+ addNewInput(inputLoc: GridPos)